COBB
INSIDE VISUAL dBASE™

November 1996 • Vol. 3 No. 11
US $8.50

Tips & techniques for Visual dBASE and dBASE for Windows

Using conditional compilation to create demo applications

I prefer to collect full payment for my programming projects up front. Who doesn't? Unfortunately, I live on Earth, where few clients are willing to pay until they've seen and played with the programs I write. I'm working on a way around this problem (the "getting clients to pay up front" part, not the "living on Earth" part), but until I come up with a solution, I'm stuck creating evaluation copies of some of my programs.

What makes a good demo? Some software houses offer slick presentations that include screen shots and descriptions of the program's features and functions. Others give a complete copy of the working program with a license to use the program for a limited period of time.

If you're like me, you may not be able to afford to create an animated, whiz-bang, slide-show-type demo. And software piracy is all too familiar—you don't want to give a fully functional copy of your program to just anyone. In my opinion, a good demo should allow potential customers to try all the features of the program, while preventing them from simply making a copy that they can use without paying for the product.

At the same time, I'm not crazy about adding unnecessarily to my already heavy workload. One solution that has worked well for me is *conditional compilation*, which allows me to produce two different executables from a single body of source code. Frequently, the only differences between a fully functioning program and a demo are a "reminder" screen and the use of sample data files.

The demo version of the program will always start with a sample data set, regardless of the data the user might have added previously while running the demo. Thus, users can add and change data to their hearts' content, produce reports, and so on. However, the next time they fire up the demo, the application will reset to the default (sample) data set, making the demo version of the application useless for any purpose other than demonstration and evaluation.

The beauty of this scheme is that you can choose the version—demo or fully functional—that you want to produce by changing a single line of code. In this article, we'll show you how to use the Visual dBASE preprocessor to perform conditional compilation with minimal effort.

Preprocessor directives: putting on the #s (pounds)

It's not unusual to see the preprocessor at work in the sample code we present in the pages of *Inside Visual dBASE*. Preprocessor directives are easy to spot because they all begin with a pound sign (#). For example, I frequently use the #define directive to create constants (data that doesn't change during the execution of the program) in my source code. Such #define constants can contain any kind of dBASE data, and you

IN THIS ISSUE

A Publication of The Cobb Group

can even construct #defines that act like procedures or functions. The most common use, though, is establishing simple constant values. For example, the command

```
#define Copyright "Copyright 1996 by Works Most
of the Time Software, Ltd."
```

establishes the copyright notice as a string constant available throughout your application. When the compiler sees this #define directive, it knows from that point forward it should insert the string *Copyright 1996 by Works Most of the Time Software, Ltd.* every time it sees the symbol *Copyright* in the source code (and outside of a quoted string) before it compiles the code.

Could you accomplish the same thing by setting a variable to the copyright notice? You bet. So why use #define? Because it's harder to make a mistake using preprocessor directives! Any part of a program can modify a global variable. However, you can't alter a #define so easily, because a #define is *not* a variable—it's an instruction to the compiler. Therefore, #defines are safer than memory variables when you're dealing with constant values.

Conditional compilation

One group of preprocessor directives tells the compiler what code to compile. When Visual dBASE compiles a program, form, query, or other source file, it normally compiles every line in the file. These directives take the form of if-else-endif blocks, just like the regular dBASE language elements with the same names. Here's a simple example:

```
#if  "Windows version 3" $ os()
   ? "Long file names are NOT available"
#else
   ? "Long file names OK"
#endif
```

In this example, only one of the statements compiles into the program, depending on the operating system version under which you compiled the code.

At first you might think the code above is identical to the following:

```
if  "Windows version 3" $ os()
   ? "Long file names are NOT available"
else
   ? "Long file names OK"
endif
```

The end result—the string printed to the screen—is indeed identical. However, the first example compiles to a smaller object or EXE file, because the #if directive tells the compiler whether or not to compile the code at all!

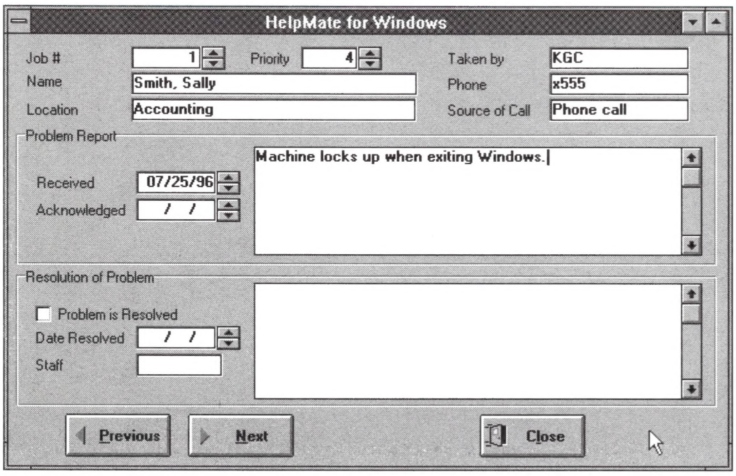
The second example also points out an important difference between the two. Specifically, the compiler evaluates #if and other preprocessor directives *at compile time*, not runtime. Once the preprocessor makes a #if decision, the result is fixed permanently in the compiled program. Because of this behavior, you should take care when making decisions based on functions like OS( ), which will vary from system to system. The compiler will permanently include the result of the #if...#else evaluation in your program, while the program will determine the result of the plain if...else evaluation at runtime. Thus, the two tests will return different results on different systems.

Let's do a demo!

To illustrate this technique, I've written a small, single-form application called HelpMate, which is a simple (okay, *very* simple) application for tracking technical support activities. Figure A shows HelpMate's main (and only) form.

I like to identify demos very clearly, usually by adding a splash screen that identifies the program as a demo or

Figure A



HelpMate for Windows

Job # 1 Priority 4 Taken by KGC

Name Smith, Sally Phone x555

Location Accounting Source of Call Phone call

Problem Report

Received 07/25/96

Acknowledged / /

Machine locks up when exiting Windows.

Resolution of Problem

□ Problem is Resolved

Date Resolved / /

Staff

Here's the HelpMate application in action.

evaluation version and advises the user of the limitations imposed on this version. If your needs are very simple, even the MsgBox( ) function could serve that purpose. However, I like to have a little more control over the look of the box, so I always create my own. In this case, I created a splash screen called DEMO.WFM. Figure B shows DEMO.WFM running, and Listing A on page 5 contains the code for the form.

As you peruse the code listing, you'll notice that I've set the Maximize, Minimize, Sizeable, and MDI properties to .F. When you launch this form, you'll use ReadModal( ) instead of Open( ), since you want the user to notice the screen. You don't want the user to minimize or maximize the form, so you'll set those properties and a couple of others to restrict the user's actions.

The essential modifications

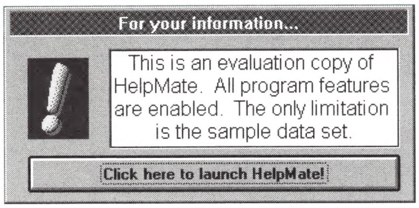
Now let's do some patchwork on HELPMATE.WFM to let you easily create a demo version of the program. Since HelpMate consists of a single form, all the action will happen in the WFM file's header. If you were developing an application that launches from a PRG program file, you'd place the code we're about to add in that file.

The header section of a WFM file created with the Form Designer normally looks like this:

```
** END HEADER * do not remove this line*
* Generated on 06/19/96
*
parameter bModal
local f
f = new HELPMATEFORM()
if (bModal)
   f.mdi = .F. && ensure not MDI
   f.ReadModal()
else
   f.Open()
endif
CLASS HELPMATEFORM OF FORM
*... and so on....
```

Note that the first line in the file is the END of the header section. In other words, the Form Designer doesn't produce any header code. The header is valuable to you as a developer, though, because you can place any code you like in that section, and the Form Designer will leave the code alone when it writes the WFM file. So, the header is the place to add custom code you need in order to properly set up the form or its operating environment. This time around, we'll use the header to set up conditional compilation for the demo version.

Figure B



This is an evaluation copy of HelpMate. All program features are enabled. The only limitation is the sample data set.

The DEMO.WFM splash screen identifies the demo version of the program.

Let's think through the steps we'll need to execute in our demo version of HelpMate. First, we want to launch the DEMO.WFM form to nag and inform the user. Second, we want to wipe out any data that might exist from previous use of the program. Third, we need to give the user some data to play with, so we'll need to rebuild the data file and restore some sample records.

If we're compiling a non-demo version of the program, we don't want to perform *any* of those tasks. This is where conditional compilation comes into play.

The #if preprocessor directive we discussed earlier works on any condition you can mold into a logical expression. For HelpMate, we'll make the preprocessor directive even simpler than that. The #ifdef directive turns on compiling if you've #defined the specified symbol. If you haven't defined the symbol, the code between the #ifdef and the next #else or #endif will be omitted from the resulting program. For example, the following code would print *"Symbol is defined!"*:

```
#define EVALUATION
#ifdef EVALUATION   && Since EVALUATION is
#defined, the following will compile....
   ? "Symbol is defined!"
#endif
```

To stop the compilation of the print statement, we need to omit or remark out the *#define EVALUATION* line. Alternately, we can #undef the symbol. The preprocessor directive #undef "undefines" a symbol so that the compiler thinks it was never #defined in the first place, as shown in the following commands:

```
#define EVALUATION
#undef EVALUATION
* Now it's as if the above #define statement
* never happened!
#ifdef EVALUATION
   ? "Symbol is defined!"
* this line will NOT be compiled this time
#endif
```

Now, let's apply this concept to HELPMATE.WFM. I've included line numbers in Figure C for easy reference. Remember, all of the following code goes in the header section of the WFM file. See Listing B for the whole file with conditional compilation in place.

In line 1, we either #define or #undef EVALUATION. If defined, the program will include lines 5 through 12 in the compilation. If not defined, the program omits these lines from the compiled program.

Figure C

```
1    #define EVALUATION
2    parameter bModal
3    local f
4    #ifdef EVALUATION
5       do DEMO.WFM with .T.  && Demo splash screen
6       use calls exclusive
7       set safety off
8       zap
9    AddRecord(1,"KGC",date(),"Smith, Sally","Accounting","x555",;
         4,"Machine locks up when exiting Windows.","Phone call")
10   AddRecord(2,"SWR",date(),"Whiner, Steve","Sales","x555",;
                   1,"Can't find the Solitaire icon","Email")
11   AddRecord(3,"JED",date(),"Worry, Wendy","Marketing","x555",;
                  2,"Can't read floppy disks from commercial printer","Email")
12      use
13   #endif
14   f = new HELPMATEFORM()
15   if (bModal)
16      f.mdi = .F. && ensure not MDI
17      f.ReadModal()
18   else
19      f.Open()
20   endif
21   Return

22   Procedure AddRecord
23     parameter jobnum,takenby,recdate,name,location,phone,;
24               priority,problem
25     append automem
26   Return

27   ** END HEADER * do not remove this line*
```

Thus, we can simply change #define to #undef or comment the #define line to switch from a "normal" to a "demo" compile.

Line 5 calls DEMO.WFM with .T. This step opens the form with ReadModal( ), as we discussed earlier. Lines 6 through 8 open and clear the file. Lines 9 through 11 call AddRecord( ), a simple procedure I attached to the header to quickly add sample records with APPEND AUTOMEM. Lines 22 through 26 contain the Procedure AddRecord.

Lines 14 through 20 might look familiar; they contain the code that's normally just below the header section of the WFM file. Since it's common practice to open forms with DO statements (as we did with DEMO.WFM in line 5), it's good form to include this code so that the WFM file will behave in a predictable way when the user double-clicks it in the dBASE Navigator or launches the WFM file with a DO command.

Finally, notice the RETURN statement after our custom code (line 21). This command ensures that the form's "regular" code doesn't run after our custom code. (In most cases, you'll almost never want this code to run, and an extra RETURN guarantees that it won't.) This command will cause the compiler to display messages that read *Warning - Line: 47 Command will never be reached*. You can safely ignore these warnings.

#define conclusion

Creating demo versions of your applications is just one way to use conditional compilation. Another common use is to trigger the inclusion or exclusion of adhoc debugging code (usually in the form of ? and MsgBox( ) statements) in critical parts of an application. Keep an eye out for opportunities; you'll find lots of uses for those little pound signs! ❖

Listing A: *DEMO.WFM*

```
** END HEADER * do not remove this line*
* Generated on 07/15/96
*
parameter bModal
local f
f = new DEMOFORM()
if (bModal)
   f.mdi = .F. && ensure not MDI
   f.ReadModal()
else
   f.Open()
endif
CLASS DEMOFORM OF FORM
   this.Text = "For your information..."
   this.EscExit = .F.
   this.Maximize = .F.
   this.Left = 26
   this.Minimize = .F.
   this.Sizeable = .F.
   this.SysMenu = .F.
   this.Top = 4.7646
   this.MDI = .F.
   this.Height = 7.4697
   this.Width = 52.666

   DEFINE TEXT TX_MESSAGE OF THIS;
   PROPERTY;
       FontBold .F.,;
       Alignment 10,;
   Text "This is an evaluation copy of HelpMate.
   All program features are enabled.
The only limitation is the sample data set.",;
         Border .T.,;
         Left 13,;
         Top 0.5879,;
         FontName "Arial",;
         Height 4.6465,;
         FontSize 12,;
         Width 38.5,;
         ColorNormal "N/W+"

   DEFINE IMAGE LOGO OF THIS;
       PROPERTY;
         Alignment 1,;
         Left 2.3311,;
         Top 1,;
         Height 4,;
         Width 8.8311,;
         DataSource "RESOURCE #128"

   DEFINE PUSHBUTTON PB_CLOSE OF THIS;
       PROPERTY;
         Group .T.,;
         Text "Click here to launch HelpMate!",;
         Left 2.3311,;
         OnClick {; form.close()},;
         Top 5.5293,;
         Height 1.7051,;
         Width 49.167

ENDCLASS
```

Listing B: *HELPMATE.WFM*

```
#define EVALUATION

parameter bModal
local f

#ifdef EVALUATION
   do demo.wfm with .T.  && Demo splash screen
   use calls excl
   set safety off
   zap
   AddRecord(1,"KGC",date(),"Smith, Sally","Accounting","x555",;
         4,"Machine locks up when exiting Windows.","Phone call")
   AddRecord(2,"SWR",date(),"Whiner, Steve","Sales","x555",;
              1,"Can't find the Solitaire icon","Email")
   AddRecord(3,"JED",date(),"Worry, Wendy","Marketing","x555",;
             2,"Can't read floppy disks from commercial printer","Email")
#endif

f = new HELPMATEFORM()
if (bModal)
   f.mdi = .F. && ensure not MDI
   f.ReadModal()
else
   f.Open()
endif

return

procedure AddRecord
  parameter jobnum,takenby,recdate,name,location,phone,;
            priority,problem
  append automem
return

** END HEADER * do not remove this line*
* Generated on 06/19/96
*
parameter bModal
local f
f = new HELPMATEFORM()
if (bModal)
   f.mdi = .F. && ensure not MDI
   f.ReadModal()
else
   f.Open()
endif
CLASS HELPMATEFORM OF FORM
   Set Procedure To C:\VDB\SAMPLES\BUTTONS.CC additive
   this.Text = "HelpMate for Windows"
   this.Left = 4.5
   this.PageNo = 0
   this.Top = 0.5293
   this.Height = 20.5293
   this.Width = 96.666
   this.View = "CALLS.QBE"

   DEFINE RECTANGLE RECTANGLE2 OF THIS;
       PROPERTY;
   Text "Resolution of Problem",;
   FontBold .F.,;
   Left 1,;
   PageNo 0,;
   Top 11.1758,;
   Height 6.5293,;
   Width 94.165

DEFINE RECTANGLE RECTANGLE1 OF THIS;
  PROPERTY;
   Text "Problem Report",;
   FontBold .F.,;
   Left 1,;
   PageNo 0,;
   Top 4.4697,;
   Height 6.5303,;
   Width 94.166

DEFINE ENTRYFIELD EF_TAKENBY OF THIS;
  PROPERTY;
   Left 72.666,;
   Top 0.5879,;
   DataLink "CALLS->TAKENBY",;
   Height 1.1172,;
   Width 18.834

DEFINE SPINBOX SB_DATEREC OF THIS;
  PROPERTY;
   Left 17.166,;
   Rangemin {12/05/95},;
   Rangemax {03/14/96},;
   Top 6.5293,;
   DataLink "CALLS->RECDATE",;
   Height 1.1172,;
   Width 14

DEFINE SPINBOX SB_DATEACK OF THIS;
  PROPERTY;
   Left 17.166,;
   Rangemin {12/05/95},;
   Rangemax {03/14/96},;
   Top 7.8232,;
   DataLink "CALLS->ACKDATE",;
   Height 1.1172,;
   Width 14

DEFINE ENTRYFIELD EF_NAME OF THIS;
  PROPERTY;
   Left 16.5,;
   Top 1.8232,;
   DataLink "CALLS->NAME",;
   Height 1.1172,;
   Width 38.5

DEFINE ENTRYFIELD EF_LOCATION OF THIS;
  PROPERTY;
   Left 16.5,;
   Top 3.0586,;
   DataLink "CALLS->LOCATION",;
   Height 1.1172,;
   Width 38.333

DEFINE ENTRYFIELD EF_PHONE OF THIS;
  PROPERTY;
   Left 72.666,;
   Top 1.8232,;
   DataLink "CALLS->PHONE",;
   Height 1.1172,;
   Width 18.834

DEFINE SPINBOX SB_PRIORITY OF THIS;
  PROPERTY;
   Left 39.333,;
   Rangemin 1,;
   Rangemax 100,;
   Top 0.5879,;
   DataLink "CALLS->PRIORITY",;
   Height 1.1172,;
   Width 10.833

DEFINE ENTRYFIELD EF_STAFF OF THIS;
  PROPERTY;
   Left 17.166,;
   Top 15.1758,;
   DataLink "CALLS->STAFF",;
   Height 1.1182,;
   Width 11.667

DEFINE SPINBOX SB_DATERES OF THIS;
  PROPERTY;
   Left 17.166,;
   Rangemin {06/19/96},;
   Rangemax {09/27/96},;
   Top 13.8818,;
   DataLink "CALLS->RESDATE",;
   Height 1.1182,;
   Width 14

DEFINE ENTRYFIELD EF_SOURCE OF THIS;
  PROPERTY;
   Left 72.666,;
   Top 3.0586,;
   DataLink "CALLS->SOURCE",;
   Height 1.1172,;
   Width 18.834

DEFINE CHECKBOX CH_RESOLVED OF THIS;
  PROPERTY;
   Text "Problem is Resolved",;
   FontBold .F.,;
   Left 3.666,;
   Top 12.7646,;
   DataLink "CALLS->RESOLVED",;
   Height 1.1172,;
   Width 26.167,;
   Group .T.

DEFINE SPINBOX SB_JOBNUM OF THIS;
  PROPERTY;
   Left 16.5,;
   Rangemin 1,;
   Rangemax 100,;
   Top 0.5879,;
   DataLink "CALLS->JOBNUM",;
   Height 1.1172,;
   Width 12.666

DEFINE TEXT TEXT1 OF THIS;
  PROPERTY;
   Text "Job #",;
   FontBold .F.,;
   Left 2.5,;
   PageNo 0,;
   Alignment 3,;
   Top 0.5879,;
   Height 1.1172,;
   Width 11.666

DEFINE TEXT TEXT2 OF THIS;
  PROPERTY;
   Text "Priority",;
   FontBold .F.,;
   Left 32.5,;
   PageNo 0,;
   Alignment 3,;
   Top 0.5879,;
   Height 1.1172,;
   Width 6
```

```
DEFINE TEXT TEXT3 OF THIS;
    PROPERTY;
      Text "Name",;
      FontBold .F.,;
      Left 2.5,;
      PageNo 0,;
      Alignment 3,;
      Top 1.7051,;
      Height 1.1182,;
      Width 11.666

DEFINE TEXT TEXT4 OF THIS;
    PROPERTY;
      Text "Location",;
      FontBold .F.,;
      Left 2.5,;
      PageNo 0,;
      Alignment 3,;
      Top 3.0586,;
      Height 1.1172,;
      Width 11.666

DEFINE TEXT TEXT5 OF THIS;
    PROPERTY;
      Text "Taken by",;
      FontBold .F.,;
      Left 59,;
      PageNo 0,;
      Alignment 3,;
      Top 0.5879,;
      Height 1.1172,;
      Width 11.666

DEFINE TEXT TEXT6 OF THIS;
    PROPERTY;
      Text "Received",;
      FontBold .F.,;
      Left 3.833,;
      PageNo 0,;
      Alignment 3,;
      Top 6.5293,;
      Height 1.1172,;
      Width 10.167

DEFINE TEXT TEXT7 OF THIS;
    PROPERTY;
      Text "Acknowledged",;
      FontBold .F.,;
      Left 3.833,;
      PageNo 0,;
      Alignment 3,;
      Top 7.8232,;
      Height 1.1172,;
      Width 13.167

DEFINE TEXT TEXT8 OF THIS;
    PROPERTY;
      Text "Date Resolved",;
      FontBold .F.,;
      Left 3.666,;
      PageNo 0,;
      Alignment 3,;
      Top 13.8818,;
      Height 1.1182,;
      Width 12.834

DEFINE TEXT TEXT9 OF THIS;
    PROPERTY;
      Text "Source of Call",;
      FontBold .F.,;
      Left 59,;
      PageNo 0,;
      Alignment 3,;
      Top 3.0586,;
      Height 1.1172,;
      Width 12

DEFINE TEXT TEXT10 OF THIS;
    PROPERTY;
      Text "Staff",;
      FontBold .F.,;
      Left 3.666,;
      PageNo 0,;
      Alignment 3,;
      Top 15.1758,;
      Height 1.1182,;
      Width 7.834

DEFINE TEXT TEXT11 OF THIS;
    PROPERTY;
      Text "Phone",;
      FontBold .F.,;
      Left 59,;
      PageNo 0,;
      Alignment 3,;
      Top 1.8232,;
      Height 1.1172,;
      Width 8

DEFINE EDITOR EDITOR1 OF THIS;
    PROPERTY;
      CUATab .T.,;
      Left 33,;
      PageNo 0,;
      Top 5.3525,;
      DataLink "CALLS->PROBLEM",;
      Height 5.1768,;
      Width 60.166

DEFINE EDITOR EDITOR2 OF THIS;
    PROPERTY;
      CUATab .T.,;
      Left 33,;
      PageNo 0,;
      Top 11.8232,;
      DataLink "CALLS->RESOLUTION",;
      Height 5.5293,;
      Width 60.166

DEFINE NEXTBUTTON NEXTBUTTON1 OF THIS;
    PROPERTY;
      Left 24.166,;
      PageNo 0,;
      Top 18.0586,;
      Height 2.0586,;
      Width 14.167,;
      Group .T.

DEFINE PREVBUTTON PREVBUTTON1 OF THIS;
    PROPERTY;
      Left 7.666,;
      PageNo 0,;
      Top 18.0586,;
      Height 2.0586,;
      Width 14.167,;
      Group .T.

DEFINE CLOSEBUTTON CLOSEBUTTON1 OF THIS;
    PROPERTY;
      Left 63.333,;
      PageNo 0,;
      Top 18.0586,;
      Height 2.0586,;
      Width 14.167,;
      Group .T.

ENDCLASS
```

EXECutive privilege

Sometimes dBASE just isn't enough. Almost every large application I've written—and many of the smaller ones—have relied on the dBASE RUN command or the Run( ) function to call external programs to handle tasks like formatting disks, setting up network printer parameters, and so on.

I've never been totally satisfied with the effects of running an external program, mainly because doing so clutters the screen. Running any garden-variety DOS program or command from within Visual dBASE, for instance, results in a really ugly screen. For example, Figure A shows the less-than-attractive result of formatting a disk using the DOS FORMAT command. Listing A, on page 10, contains the listing for the Format A Floppy Diskette form.

The form uses the dBASE Run( ) function. Run( ) uses the settings in the file DBASEWIN.PIF to determine the characteristics of the DOS window the command will run in. (Compiled applications use DB55RUN.PIF.) Unfortunately, you don't have much control over how your screen looks—there's precious little you can specify about the window besides whether the command will run in a window or in full-screen mode. There's no "Run Minimized" option, like the one available for Program Manager icons. In this article, we'll show you how to exercise control over your screen aesthetics by running programs with a little help from WinExec( ) and a PIF file.

PIF, PIF, hooray! (Not.)

You can create and edit PIF files with the Windows PIFEDIT program. Figure B shows the PIFEDIT program editing the DBASEWIN.PIF file, which simply calls COMMAND.COM. When you use the Run( ) function or RUN command in a program, any parameters pass to the PIF file on the command line. In FORMAT.WFM, the line that passes those parameters reads:

```
Run(.f.,"FORMAT "+form.cb_drives.value+" /U
➥ /BACKUP /V:None")
```

The first parameter specifies whether the program to run is a Windows (.T.) or DOS (.F.) application. The second parameter is the command that COMMAND.COM will execute in the DOS shell.

WinExec( ) to the rescue

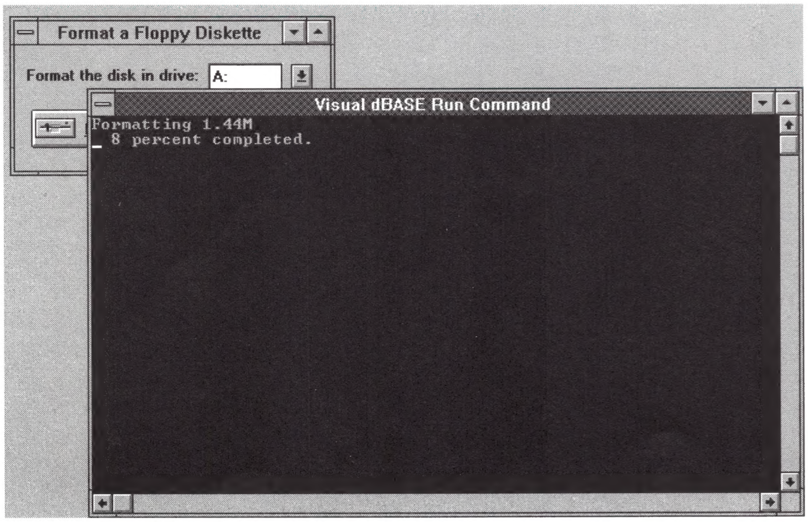
Neither PIF files nor the Run( ) function let you run a program minimized. However, the Windows API function WinExec does. The WINAPI.H file that ships with Visual dBASE lists this function. The entry in WINAPI.H reads as follows:

```
extern  CINT  WinExec ( CSTRING,CINT )  KERNEL
```

You'll need to include this line in any program that uses the WinExec( ) function. The first argument is a string containing the program (and any command-line arguments) to run. The second argument is a number specifying the "ShowWindow" status. The WINAPI.H file includes the values for ShowWindow, but lists them as "ShellExecute( ) Window display options." (The WINAPI.H file's comments don't explicitly mention this fact.)

The ShellExecute API function executes programs and also performs other tasks. It's a more complicated function than WinExec( ), and in fact isn't even listed by name in the WINAPI.H file, so we'll stick with the simpler WinExec( ) function. Here are the values for the

Figure A



```
Formatting 1.44M
_ 8 percent completed.
```

The FORMAT utility is hogging the screen.

ShowWindow status as #defined in the WINAPI.H file:

```
#define SW_HIDE             0
#define SW_MAXIMIZE         3
#define SW_MINIMIZE         6
#define SW_NORMAL           1
#define SW_RESTORE          9
#define SW_SHOW             5
#define SW_SHOWMAXIMIZED    3
#define SW_SHOWMINIMIZED    2
#define SW_SHOWMINNOACTIVE  7
#define SW_SHOWNA           8
#define SW_SHOWNOACTIVATE   4
#define SW_SHOWNORMAL       1
```

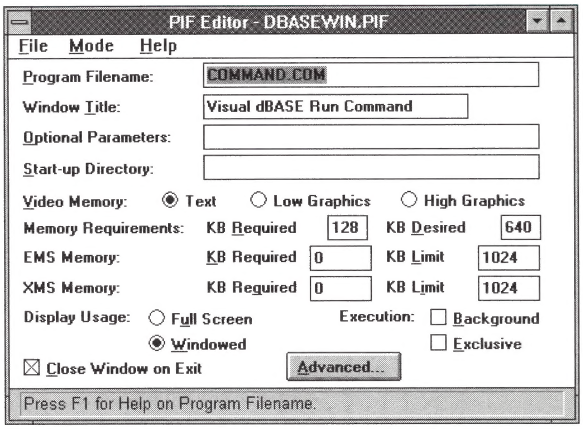
SW_HIDE and SW_SHOWMINIMIZED provide just the functionality we need. So, let's learn a bit more about WinExec( ).

First, WinExec( ) is pickier than Run( ) about the program you pass to it. While Run( ) will find and run any executable in the DOS PATH, you need to tell WinExec exactly where to find the program.

This requirement is easy to meet if the program you're executing is a PIF file of your own creation, and that's exactly what we'll do. Figure C shows the settings I use for my custom PIF file, DOSSHELL.PIF. I usually come up with something more snazzy than *DOS Shell* for a Window title—be creative (or at least use a title that makes sense!) if end users are going to see this window.

Next, we need to know if something goes wrong. The CINT (a C language integer value that dBASE interprets as a simple number) that WinExec( ) returns is either a Windows handle to the process it created when the program executed, or a value less than 32 if a problem occurred. These values aren't listed in WINAPI.H, so we've provided a reference list in Table A.

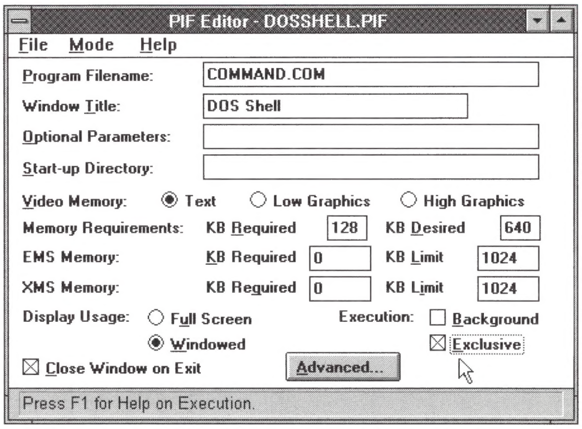
Figure B



Program Filename: COMMAND.COM
Window Title: Visual dBASE Run Command
Optional Parameters:
Start-up Directory:
Video Memory: ● Text ○ Low Graphics ○ High Graphics
Memory Requirements: KB Required 128 KB Desired 640
EMS Memory: KB Required 0 KB Limit 1024
XMS Memory: KB Required 0 KB Limit 1024
Display Usage: ○ Full Screen ● Windowed
Execution: □ Background □ Exclusive
☒ Close Window on Exit
Press F1 for Help on Program Filename.

You use PIFEDIT to create and modify Windows Program Information Files, although it doesn't offer much help with window position or size.

Figure C



Program Filename: COMMAND.COM
Window Title: DOS Shell
Optional Parameters:
Start-up Directory:
Video Memory: ● Text ○ Low Graphics ○ High Graphics
Memory Requirements: KB Required 128 KB Desired 640
EMS Memory: KB Required 0 KB Limit 1024
XMS Memory: KB Required 0 KB Limit 1024
Display Usage: ○ Full Screen ● Windowed
Execution: □ Background ☒ Exclusive
☒ Close Window on Exit
Press F1 for Help on Execution.

We used PIFEDIT to create a generic DOSSHELL.PIF file for use with WinExec( ).

Table A: *Error status values returned by WinExec( )*

0	System was out of memory, executable file was corrupt, or relocations were invalid.
2	File was not found.
3	Path was not found.
5	Attempt was made to dynamically link to a task, or there was a sharing or network-protection error.
6	Library required separate data segments for each task.
8	There was insufficient memory to start the application.
10	Windows version was incorrect.
11	Executable file was invalid. Either it was not a Windows application or there was an error in the EXE image.
12	Application was designed for a different operating system.
13	Application was designed for MS-DOS 4.0.
14	Type of executable file was unknown.
15	Attempt was made to load a real-mode application (developed for an earlier version of Windows).
16	Attempt was made to load a second instance of an executable file containing multiple data segments not marked read-only.
19	Attempt was made to load a compressed executable file. The file must be decompressed before it can be loaded.
20	Dynamic link library (DLL) file was invalid. One of the DLLs required to run this application was corrupt.
21	Application requires 32-bit extensions.

Gluing the pieces together

Now that we've got a PIF file that calls COMMAND.COM and takes any old arguments we want to throw at it, let's combine it with WinExec( ) to get the desired effect. We'll go back to FORMAT.WFM and make the change in the PB_FORMAT_ONCLICK procedure, replacing the Run( ) function with our new WinExec( ) call. We'll also EXTERN WinExec if needed, as follows:

```
Procedure PB_FORMAT_OnClick
   local nStatus
   if type("WinExec") # "FP"
      extern  CINT WinExec (CSTRING,CINT) KERNEL
      #define SW_SHOWMINIMIZED 2
   endif
   nStatus = WinExec("DOSSHELL.PIF /C FORMAT
➥      "+form.cb_drives.value+" /U /BACKUP
➥      /V:None",SW_SHOWMINIMIZED)
   if nStatus < 32
      MsgBox("Error "+ltrim(str(nStatus))+"
➥         occurred during WinExec() call","Oops!")
   endif
```

Depending on the program you run when you use this technique, you might want to use SW_HIDE or one of the other ShowWindow values.

A new way to Run( )

Now that we've seen a simple example of using WinExec( ) with a PIF file, it's easy to go one more step and create a generic function that you can use in any application. Listing B, WinRun( ), is an example of such a function. It handles the errors that might occur, and reduces the argument for Window Status to a simple logical value: .T. to display the window normally or .F. to minimize it. The default will be .T. if you omit the Status argument. You can easily modify or extend this function to meet your own needs. ❖

Listing A: *FORMAT.WFM*

```
** END HEADER * do not remove this line*
* Generated on 06/19/96
*
parameter bModal
local f
f = new FORMATFORM()
if (bModal)
   f.mdi = .F. && ensure not MDI
   f.ReadModal()
else
   f.Open()
endif
CLASS FORMATFORM OF FORM
   this.Text = "Format a Floppy Diskette"
   this.Left = 10.5
   this.Top = 1.6465
   this.Height = 5.8818
   this.Width = 41.666
   this.mdi = .F.

   DEFINE COMBOBOX CB_DRIVES OF THIS;
       PROPERTY;
         OnOpen CLASS::CB_DRIVES_ONOPEN,;
         Left 25.666,;
         Sorted .T.,;
         DataSource 'ARRAY {"A:","B:"}',;
         Top 0.8818,;
         Height 1.1768,;
         Width 13.667,;
         Style 1

   DEFINE TEXT TEXT1 OF THIS;
       PROPERTY;
         Text "Format the disk in drive:",;
         Left 1.5,;
         Top 1,;
         Height 1.1172,;
         Width 23.666

   DEFINE PUSHBUTTON PB_FORMAT OF THIS;
       PROPERTY;
         Text "&Format it!",;
         UpBitmap "RESOURCE #2170",;
         Left 2.166,;
         OnClick CLASS::PB_FORMAT_ONCLICK,;
         Top 3,;
         Height 1.8232,;
         Width 17.834,;
         Group .T.

   DEFINE PUSHBUTTON PB_CANCEL OF THIS;
       PROPERTY;
         Text "&Cancel",;
         UpBitmap "RESOURCE #28",;
         Left 21,;
         OnClick CLASS::PB_CANCEL_ONCLICK,;
         Top 3,;
         Height 1.8232,;
         Width 17.833,;
         Group .T.

   Procedure CB_DRIVES_OnOpen
      this.value = "A:"

   Procedure PB_CANCEL_OnClick
      form.close()

   Procedure PB_FORMAT_OnClick
      Run(.f.,"FORMAT "+form.cb_drives.value+" /U
➥        /BACKUP /V:None")

ENDCLASS
```

Listing B: *WRUN.PRG*

```
* WRun(cProgram,lStatus): Run a program with Normal or Minimized WindowState
* cProgram is a string specifying the program and any required arguments
* lStatus is .T. for normal window, .F. for minimized. Default is .T.
*
* Examples:  x = WRun("DIR >> DIR.LST",.F.)  && runs the command minimized.
*           x = WRun("TREE ¦ PAUSE",.T.)  && runs normally.
*
Procedure WRun
   parameters cProgram, lShow
   local nStatus, nShow
   #define SW_SHOW            5
   #define SW_SHOWMAXIMIZED   3
   #define SW_SHOWMINIMIZED   2

   if type("cProgram") # "C"
      return
   endif
   if type("lShow") # "L"
      lShow = .T.
   endif

   if type("WinExec") # "FP"
      extern  CINT WinExec (CSTRING,CINT) KERNEL
   endif
   nShow = iif(lShow,SW_SHOW,SW_SHOWMINIMIZED)
   nStatus = WinExec("DOSSHELL.PIF /C " + cProgram,nShow)
   if nStatus < 32
      aErrors = {"System was out of memory, executable file was corrupt, or relocations were invalid.",;
           "","File was not found.","Path was not found.","",;
           "Attempt was made to dynamically link to a task, or there was a sharing or network-protection error.",;
           "Library required separate data segments for each task.",;
           "","There was insufficient memory to start the application.",;
           "", "Windows version was incorrect.",;
           "Executable file was invalid. Either it was not a Windows application or there was an error in the EXE image.",;
           "Application was designed for a different operating system.",;
           "Application was designed for MS-DOS 4.0.",;
           "Type of executable file was unknown. ",;
           "Attempt was made to load a real-mode application (developed for an earlier version of Windows).",;
           "Attempt was made to load a second instance of an executable file containing multiple data
➥               segments that were not marked read-only.",;
           "","","Attempt was made to load a compressed executable file. The file must be decompressed before it can be loaded.",;
           "Dynamic link library (DLL) file was invalid. One of the DLLs required to run this application was corrupt.",;
           "Application requires 32-bit extensions."}
      MsgBox(aError[nStatus],"An error occurred calling WinExec()")
   endif
return nStatus
```

WORKING SMARTER

Establishing good relations

The cornerstone of a relational database development system like dBASE is its ability to relate two or more tables in a pertinent way. Borland designed Visual dBASE to encourage the developer to use the Query Builder to establish relations between tables. It's important to understand some of the workings that go on "under the hood" of a query, since there are situations for which the Query Designer isn't optimized.

In this article, we'll cover the hows and whys of dBASE's SET RELATION command, and we'll examine alternative mechanisms for linking data from multiple tables together. Finally, we'll discuss cases in which you probably *don't* want to use SET RELATION, even though it seems the obvious way to go.

A brief relational history

Long ago, far away, dBASE II supported two open tables and a simple SET LINKAGE command to join the two. We've come a long way, baby! Visual dBASE supports multiple sessions, each sporting 225 available work areas for our tables, and so much more. As you might imagine, SET RELATION has had to "grow up" a bit to keep pace. And, boy, how it's grown: Visual dBASE introduces many new additions to SET RELATION that weren't available in dBASE for DOS.

Start simple

Fortunately, you can still do a simple relation, well, simply! For the following examples, I'll use the CUSTOMER.DBF, ORDERS.DBF, and LINEITEM.DBF tables that Visual dBASE installs in the SAMPLES directory.

Our first example is a relation between the CUSTOMER and ORDERS tables with which we can produce a list of customers and orders they've placed. (How's that for obvious?) First, let's take a look at the structure of CUSTOMER and ORDERS.

The CUSTOMER table stores essential information about a customer. Note that this information doesn't include any details about transactions from the customer's account. Good design dictates that such data should be in their own tables, and we'll see that they are. Here's the structure of the CUSTOMER table:

```
Structure for table    D:\VDB\SAMPLES\CUSTOMER.DBF
Table type             DBASE
Number of records      55
Last update            06/29/94

Field  Field Name          Type          Length   Dec   Index
    1  CUSTOMER_N          CHARACTER     4              Y
    2  NAME                CHARACTER     30             N
    3  STREET              CHARACTER     30             N
    4  CITY                CHARACTER     15             N
    5  STATE_PROV          CHARACTER     20             N
    6  ZIP_POSTAL          CHARACTER     10             N
    7  COUNTRY             CHARACTER     20             N
    8  PHONE               CHARACTER     15             N
    9  FIRST_CONT          DATE          8              N
   10  YTD_SALES           NUMERIC       12       2     N
   11  CREDIT_OK           LOGICAL       1              N
   12  SIGNATURE           BINARY        10             N
   13  NOTES               MEMO          10             N

** Total **                              186
```

The ORDERS table contains the header information for each order placed. Note that the only information ORDERS and CUSTOMER share is the CUSTOMER_N (customer number) field. The ORDERS table uses this structure:

```
Structure for table    D:\VDB\SAMPLES\ORDERS.DBF
Table type             DBASE
Number of records      83
Last update            06/13/94

Field  Field Name      Type           Length   Dec   Index
    1  ORDER_NO        CHARACTER      4              Y
    2  CUSTOMER_N      CHARACTER      4              Y
    3  SALE_DATE       DATE           8              N
    4  SHIP_DATE       DATE           8              N
    5  SHIP_VIA        CHARACTER      7              N
    6  AMT_PAID        NUMERIC        9        2     N
    7  TERMS           CHARACTER      6              N
    8  PAY_METHOD      CHARACTER      7              N
    9  TOTAL           NUMERIC        10             N

** Total **                           64
```

To produce our list of customers with the orders they've placed, we need to tie the two tables together in a meaningful way. The CUSTOMER_N field is our only choice, since there's no other data common to both tables.

A simple SET RELATION command will establish the link between the two. Remember to change to the Visual dBASE SAMPLES directory. Then, you can enter the following commands in the Command Window or in a PRG file for future reference:

```
use customer in select()
use orders order customer_n in select()
select customer
set relation to customer_n into orders
browse fields customer->customer_n,customer->name,;
orders->sale_date,orders->amt_paid
```

Figure A shows the resulting BROWSE window. Looks great, doesn't it? Well, not quite. Each record in the BROWSE window details only a single entry from the ORDERS table, and that's rarely the data we're actually interested in. This behavior is a problem that's unique to BROWSE and BROWSE controls. For most programming situations and reports, the simple relation we've set up here will provide the linkage we need between tables.

Now let's make the BROWSE window's data more complete. Close the BROWSE window, execute a SET SKIP command, then invoke the BROWSE again:

```
set skip to orders
browse fields customer->customer_n,customer->name,;
orders->sale_date,orders->amt_paid
```

SET SKIP tells dBASE to move the record pointer in the specified table through all records that match the key value in the parent table (CUSTOMER, in our example) before advancing the record pointer in the parent. The result is shown in Figure B. As you can see, this BROWSE window is more useful, since it allows us to view every order placed for a given customer.

In any relation, you must index the child table (ORDERS, in this example) on the field used for the relation. Note that we didn't have to set any particular order for the parent (CUSTOMER) table, though. We're free to order the parent table any way we like; for instance, we could order a report by customer name, ZIP code, or any other index we might create.

Multiple relations

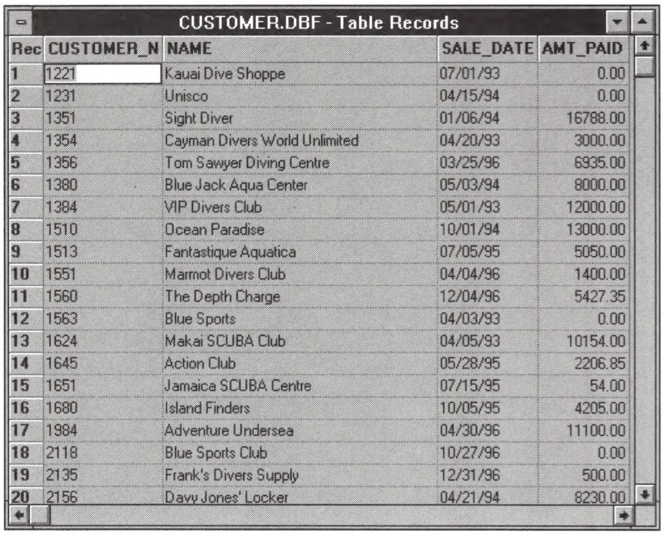
Setting up relations between more than two tables is just as simple. Let's add the LINEITEM sample table to our example so we can see what items we included in each order. Here's the structure of the LINEITEM table:

```
Master index: CUSTOMER_N
Master index: ORDER_NO
Structure for table      C:\VDB\SAMPLES\LINEITEM.DBF
Table type               DBASE
Number of records        343
Last update              06/10/94

Field  Field Name        Type          Length     Dec     Index
    1  ORDER_NO          CHARACTER     4                  Y
    2  STOCK_NO          CHARACTER     5                  N
    3  SELL_PRICE        NUMERIC       8          2       N
    4  QTY               NUMERIC       4                  N

** Total **                            22
```

The LINEITEM table has nothing in common with our CUSTOMER table, but it does have ORDER_NO in common with the ORDERS table. So, let's start with a clean slate and set up two relations—one between CUSTOMER and ORDERS, and another between ORDERS and LINEITEM, as follows:

```
clear all
use customer in select()
use orders order customer_n in select()
use lineitem order order_no in select()
select customer
set relation to customer_n into orders
select orders
set relation to order_no into lineitem
select customer
browse fields customer->customer_n, customer->name,;
orders->sale_date, orders->amt_paid, ;
lineitem->stock_no,lineitem->sell_price,lineitem->qty
```

Figure A

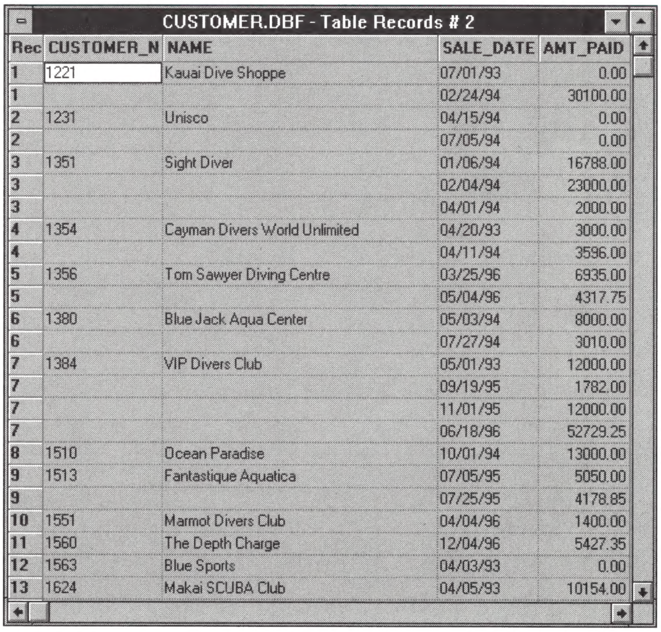


CUSTOMER.DBF - Table Records

Rec	CUSTOMER_N	NAME	SALE_DATE	AMT_PAID
1	1221	Kauai Dive Shoppe	07/01/93	0.00
2	1231	Unisco	04/15/94	0.00
3	1351	Sight Diver	01/06/94	16788.00
4	1354	Cayman Divers World Unlimited	04/20/93	3000.00
5	1356	Tom Sawyer Diving Centre	03/25/96	6935.00
6	1380	Blue Jack Aqua Center	05/03/94	8000.00
7	1384	VIP Divers Club	05/01/93	12000.00
8	1510	Ocean Paradise	10/01/94	13000.00
9	1513	Fantastique Aquatica	07/05/95	5050.00
10	1551	Marmot Divers Club	04/04/96	1400.00
11	1560	The Depth Charge	12/04/96	5427.35
12	1563	Blue Sports	04/03/93	0.00
13	1624	Makai SCUBA Club	04/05/93	10154.00
14	1645	Action Club	05/28/95	2206.85
15	1651	Jamaica SCUBA Centre	07/15/95	54.00
16	1680	Island Finders	10/05/95	4205.00
17	1984	Adventure Undersea	04/30/96	11100.00
18	2118	Blue Sports Club	10/27/96	0.00
19	2135	Frank's Divers Supply	12/31/96	500.00
20	2156	Davy Jones' Locker	04/21/94	8230.00

The first ORDERS record for each customer appears, along with data from the CUSTOMER table.

Figure B



CUSTOMER.DBF - Table Records # 2

Rec	CUSTOMER_N	NAME	SALE_DATE	AMT_PAID
1	1221	Kauai Dive Shoppe	07/01/93	0.00
1			02/24/94	30100.00
2	1231	Unisco	04/15/94	0.00
2			07/05/94	0.00
3	1351	Sight Diver	01/06/94	16788.00
3			02/04/94	23000.00
3			04/01/94	2000.00
4	1354	Cayman Divers World Unlimited	04/20/93	3000.00
4			04/11/94	3596.00
5	1356	Tom Sawyer Diving Centre	03/25/96	6935.00
5			05/04/96	4317.75
6	1380	Blue Jack Aqua Center	05/03/94	8000.00
6			07/27/94	3010.00
7	1384	VIP Divers Club	05/01/93	12000.00
7			09/19/95	1782.00
7			11/01/95	12000.00
7			06/18/96	52729.25
8	1510	Ocean Paradise	10/01/94	13000.00
9	1513	Fantastique Aquatica	07/05/95	5050.00
9			07/25/95	4178.85
10	1551	Marmot Divers Club	04/04/96	1400.00
11	1560	The Depth Charge	12/04/96	5427.35
12	1563	Blue Sports	04/03/93	0.00
13	1624	Makai SCUBA Club	04/05/93	10154.00

The SET SKIP command forces all child records to appear with their parent record.

We've broken the BROWSE command across several lines for cosmetic purposes only; you'd normally type it all in one line. Figure C on the following page shows the results; we can now see fields from all three tables.

Next we need to add SET SKIP to the mix again, but this time we'll apply it to both the ORDERS and LINEITEM tables. Close the BROWSE window and add the following code:

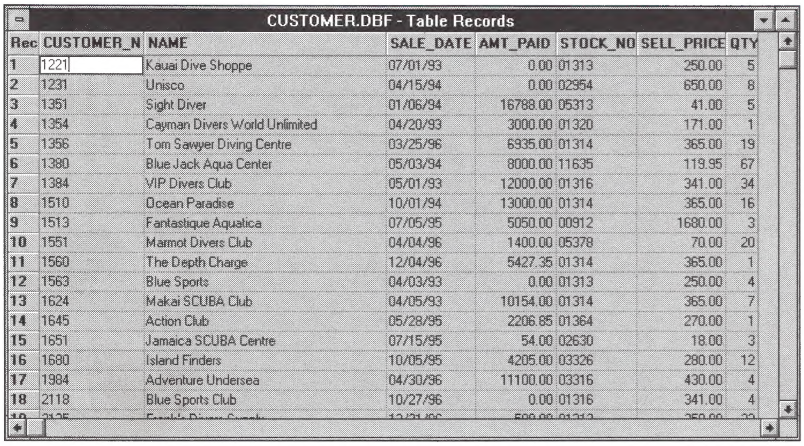
```
set skip to orders,lineitem
browse fields customer->customer_n,;
customer->name,orders->sale_date,;
orders->amt_paid,lineitem->stock_no,;
lineitem->sell_price,lineitem->qty
```

Much better! As Figure D illustrates, we can now use BROWSE to see the items each customer ordered.

It's just an expression

So far, all the relations we've set have centered around the use of simple key fields. You can use compound keys as well—in fact, you can use almost any valid expression as the basis for a relation. You must take care to get the expression correct, of course. Here's a simple example you can try with the sample tables:

Figure C

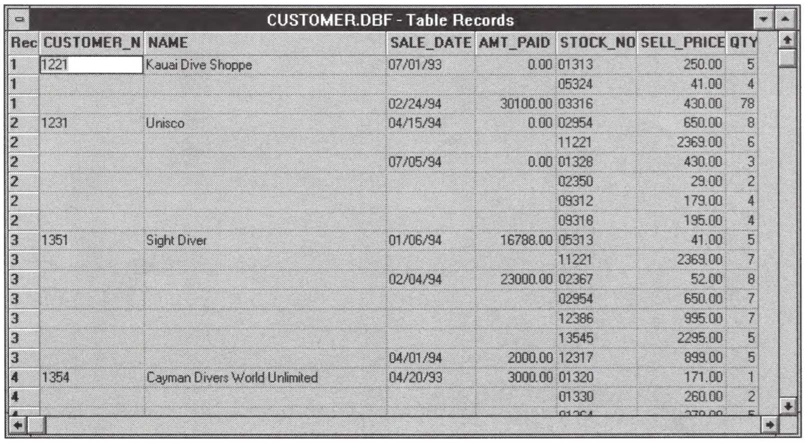


CUSTOMER.DBF - Table Records

Rec	CUSTOMER_N	NAME	SALE_DATE	AMT_PAID	STOCK_NO	SELL_PRICE	QTY
1	1221	Kauai Dive Shoppe	07/01/93	0.00	01313	250.00	5
2	1231	Unisco	04/15/94	0.00	02954	650.00	8
3	1351	Sight Diver	01/06/94	16788.00	05313	41.00	5
4	1354	Cayman Divers World Unlimited	04/20/93	3000.00	01320	171.00	1
5	1356	Tom Sawyer Diving Centre	03/25/96	6935.00	01314	365.00	19
6	1380	Blue Jack Aqua Center	05/03/94	8000.00	11635	119.95	67
7	1384	VIP Divers Club	05/01/93	12000.00	01316	341.00	34
8	1510	Ocean Paradise	10/01/94	13000.00	01314	365.00	16
9	1513	Fantastique Aquatica	07/05/95	5050.00	00912	1680.00	3
10	1551	Marmot Divers Club	04/04/96	1400.00	05378	70.00	20
11	1560	The Depth Charge	12/04/96	5427.35	01314	365.00	1
12	1563	Blue Sports	04/03/93	0.00	01313	250.00	4
13	1624	Makai SCUBA Club	04/05/93	10154.00	01314	365.00	7
14	1645	Action Club	05/28/95	2206.85	01364	270.00	1
15	1651	Jamaica SCUBA Centre	07/15/95	54.00	02630	18.00	3
16	1680	Island Finders	10/05/95	4205.00	03326	280.00	12
17	1984	Adventure Undersea	04/30/96	11100.00	03316	430.00	4
18	2118	Blue Sports Club	10/27/96	0.00	01316	341.00	4

The LINEITEM table provides details on each order. Or does it?

Figure D



CUSTOMER.DBF - Table Records

Rec	CUSTOMER_N	NAME	SALE_DATE	AMT_PAID	STOCK_NO	SELL_PRICE	QTY
1	1221	Kauai Dive Shoppe	07/01/93	0.00	01313	250.00	5
1					05324	41.00	4
1			02/24/94	30100.00	03316	430.00	78
2	1231	Unisco	04/15/94	0.00	02954	650.00	8
2					11221	2369.00	6
2			07/05/94	0.00	01328	430.00	3
2					02350	29.00	2
2					09312	179.00	4
2					09318	195.00	4
3	1351	Sight Diver	01/06/94	16788.00	05313	41.00	5
3					11221	2369.00	7
3			02/04/94	23000.00	02367	52.00	8
3					02954	650.00	7
3					12386	995.00	7
3					13545	2295.00	5
3			04/01/94	2000.00	12317	899.00	5
4	1354	Cayman Divers World Unlimited	04/20/93	3000.00	01320	171.00	1
4					01330	260.00	2

Adding a SET SKIP command to ORDERS and LINEITEM reveals all details of both tables, yielding a more useful display.

```
clear all
use orders exclusive
index on customer_n+upper(pay_method)+;
dtos(sale_date) to cpaydate
use customer in select()
select customer
set relation to customer_n into orders
SET EXACT OFF
set skip to orders
browse fields customer->customer_n,;
customer->name,orders->pay_method,;
orders->sale_date,orders->total
```

In this example, we index the ORDERS table on an expression that combines the CUSTOMER_N field (which we still need in order to maintain *some* kind of connection!), PAY_METHOD, and SALE_DATE. To make this index work, we must SET EXACT OFF, since the match between the parent table's CUSTOMER_N field will *never* match the entire index expression. You can also use LEFT( ), SUBSTR( ), and similar functions to match on a piece of a complex key. (As you experiment with this technique, you'll note that it's nearly impossible to create this kind of relation with the Query Designer, as we'll discuss later in the article.)

Two-timing relations

A single parent table can be related to more than one child. In the example shown above, CUSTOMER was the parent, ORDERS was a child to CUSTOMER and a parent to LINEITEM, and LINEITEM was a child to ORDERS.

But let's turn our needs around just a bit. To produce an audit trail of all orders placed, we'd need to make ORDERS the parent, and both CUSTOMER and LINEITEM children of ORDERS. The SET RELATION command allows us to do just that:

```
clear all
use orders order datename in select()
use customer order customer_n in select()
use lineitem order order_no in select()
select orders
set relation to customer_n into customer,
order_no into lineitem
set skip to customer,lineitem
browse fields sale_date,order_no, total,;
customer->name,lineitem->stock_no,lineitem->qty
```

Note that the SET RELATION command specifies two relations, one for CUSTOMER and another for LINEITEM. The ORDERS table is now parent to both. You can even add relations after the fact; dBASE now supports an ADDITIVE clause in the SET RELATION command, so you can build relations dynamically, even at runtime.

Maintain your integrity

Since dBASE for Windows 5.0, SET RELATION has supported some new capabilities for preventing unwanted orphan records. An *orphan* is a record in a child table whose parent record has been deleted. Older versions of dBASE couldn't automatically prevent this situation; your programs had to check for the existence of parent and/or child records as related tables were updated. Let's examine these new capabilities.

CONSTRAIN

The CONSTRAIN clause prevents processing of child records without matching parents (a situation you want to avoid). Most often in my applications, I find I'm more interested in restricting processing in the other direction. In other words, I want to see only parent records that have at least one corresponding child record. SET RELATION doesn't handle this requirement, but SET FILTER does. Look at the following code:

```
use customer in select()
use orders order customer_n in select()
select customer
set relation to customer_n into orders
set skip to orders
SET FILTER TO FOUND("ORDERS")
browse fields customer->customer_n,customer->name,;
orders->sale_date,orders->amt_paid
```

The FOUND( ) function returns a true (.T.) value when dBASE finds a matching record in the child table. Customer records without a match in the child table won't be displayed.

INTEGRITY

The INTEGRITY clause instructs dBASE to automatically update the key field value in the parent table when you execute an APPEND, APPEND BLANK, or INSERT command. INTEGRITY's behavior also covers APPENDs that occur in a BROWSE or EDIT screen.

INTEGRITY also adds a level of protection against creating orphan records. For example, we can set up a relation with INTEGRITY set, as follows:

```
clear all
use orders order customer_n in select()
use customer order customer_n in select()
sele orders
set relation to customer_n into customer integrity
brow fields customer->customer_n,customer->name,;
orders->order_no,orders->customer_n
```

Note that we're displaying the CUSTOMER_N field from both the parent and child tables for this example. Now change the key value of the first record's CUSTOMER_N field, then try to navigate off the record by pressing the down arrow key. **Figure E** shows the result.

INSIDE VISUAL dBASE™

Inside Visual dBASE (ISSN 1084-1970) is published monthly by The Cobb Group.

Staff:
Contributing Editor-in-Chief .... Keith G. Chuvala
Associate Editors-in-Chief .... Jeff E. Davis
Tiffany M. Taylor
Publications Coordinator .... Maureen Spencer
Editors .... Laura Merrill
Joan McKim
Elisabeth Pehlke
Production Artist .... Alison Schwarz
Product Group Manager .... Mike Stephens
Circulation Manager .... Mike Schroeder
Associate Publisher .... Mark Kimbell
VP/Publisher .... Mark Crane
President .... John A. Jenkins

Address:
Please send tips, special requests, and other correspondence to
The Editor, *Inside Visual dBASE*
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220
E-mail address: visual_dbase_win@merlin.cobb.zd.com

For subscriptions, fulfillment questions, and requests for group subscriptions, address your letters to
Customer Relations
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220
E-mail address: customer_relations@merlin.cobb.zd.com

Phone:
Toll free US .... (800) 223-8720
Toll free UK .... (0800) 961897
Local .... (502) 493-3300
Customer Relations Fax .... (502) 491-8050
Editorial Department Fax ..... (502) 491-3433

Back Issues:
To order back issues, call Customer Relations at (800) 223-8720. Back issues cost $8.50 each, $8.95 outside the US. You can pay with MasterCard, VISA, Discover, or American Express, or we can bill you.

Postmaster:
Periodicals Postage Paid in Louisville, KY.
Postmaster: Send address changes to:
Inside Visual dBASE
P.O. Box 35160
Louisville, KY 40232

Advertising:
For information about advertising in Cobb Group journals, contact Tracee Bell Troutt at (800) 223-8720, ext. 430.

Prices:
Domestic .... $79/yr ($8.50 each)
Outside US .... $89/yr ($8.95 each)

Borland Technical Support
Technical Support
Information Line: (800) 523-7070
TechFAX System: (800) 822-4269

dBASE recognizes that changing the key value would result in orphan records, and it lets us select "Yes" to delete all the child records matching the old key value. If we choose "No," dBASE will change the key value in the parent back to its old value. This can be a very important data safety feature in your applications, especially when BROWSE commands or Browse controls expose your data to arbitrary changes by an errant user.

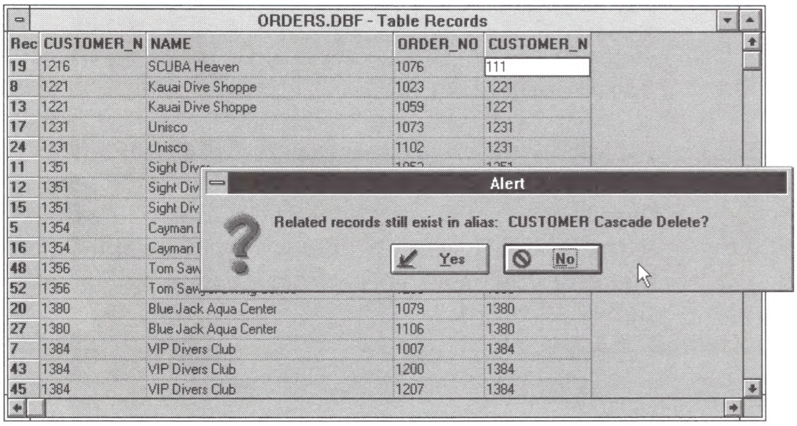
INTEGRITY CASCADE

Adding CASCADE to the INTEGRITY clause instructs dBASE to go ahead and delete child records that would become orphans when we change a key value. In other words, we're telling dBASE to assume we'd always answer "Yes" to the *Cascade delete?* question. Use this feature with care!

INTEGRITY RESTRICT

While the CASCADE addition tells dBASE to always delete records that would become orphans, RESTRICT tells dBASE to *never* delete those records. Instead, the program issues a warning message, as shown in **Figure F**, and resets the potentially offending key field to its original value.

Figure E



ORDERS.DBF - Table Records

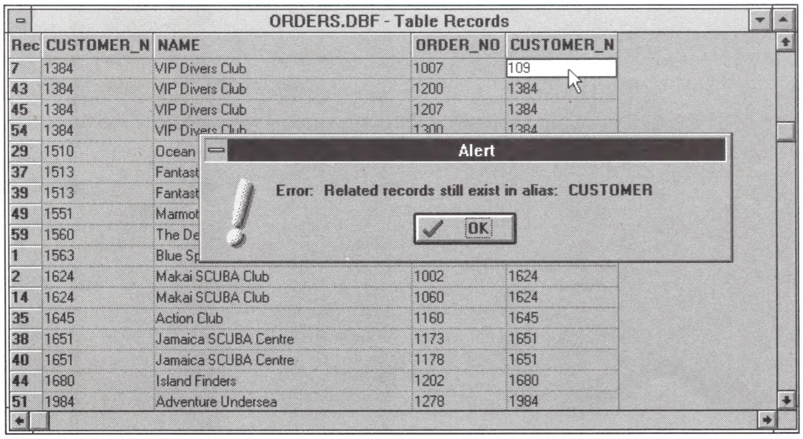
Rec	CUSTOMER_N	NAME	ORDER_NO	CUSTOMER_N
19	1216	SCUBA Heaven	1076	111
8	1221	Kauai Dive Shoppe	1023	1221
13	1221	Kauai Dive Shoppe	1059	1221
17	1231	Unisco	1073	1231
24	1231	Unisco	1102	1231
11	1351	Sight Div		
12	1351	Sight Div		
15	1351	Sight Div		
5	1354	Cayman I		
16	1354	Cayman I		
48	1356	Tom Saw		
52	1356	Tom Saw		
20	1380	Blue Jack Aqua Center	1079	1380
27	1380	Blue Jack Aqua Center	1106	1380
7	1384	VIP Divers Club	1007	1384
43	1384	VIP Divers Club	1200	1384
45	1384	VIP Divers Club	1207	1384

Alert

Related records still exist in alias: CUSTOMER Cascade Delete?

The SET RELATION...INTEGRITY option helps prevent orphan records automatically.

Figure F



ORDERS.DBF - Table Records

Rec	CUSTOMER_N	NAME	ORDER_NO	CUSTOMER_N
7	1384	VIP Divers Club	1007	109
43	1384	VIP Divers Club	1200	1384
45	1384	VIP Divers Club	1207	1384
54	1384	VIP Divers Club	1300	1384
29	1510	Ocean		
37	1513	Fantast		
39	1513	Fantast		
49	1551	Marmot		
59	1560	The De		
1	1563	Blue Sp		
2	1624	Makai SCUBA Club	1002	1624
14	1624	Makai SCUBA Club	1060	1624
35	1645	Action Club	1160	1645
38	1651	Jamaica SCUBA Centre	1173	1651
40	1651	Jamaica SCUBA Centre	1178	1651
44	1680	Island Finders	1202	1680
51	1984	Adventure Undersea	1278	1984

Alert

Error: Related records still exist in alias: CUSTOMER

You'll see this warning message when you try to delete a record after implementing the INTEGRITY RESTRICT option.

The Query Designer

You can accomplish much (perhaps all) that we've done in this article with dBASE's Query Designer (QD). The QD does indeed make a nice workshop for learning how you can establish relations. But it doesn't always use the clearest syntax, and it's fickle about using work area numbers (SELECT 1) instead of aliases (SELECT CUSTOMER). Furthermore, the QD insists on inserting a SET EXACT ON command at the top of every query, which makes "fuzzy" matches and the use of compound (multi-field) keys difficult or impossible.

The QD also has an almost obscene fondness for SET FIELDS statements, which are really useful only for end-user work. Developers should programmatically control the fields that applications display. There are suspicions of problems with SET FIELDS, and so it's probably a good idea to avoid these statements anyway (though I must admit, I've never encountered a problem with the command myself).

The good news is that you can save in a QBE file any relations you write manually; then, you can use the QBE file as the basis for a Form's View property. So, set up your relations—even test them in the Command window as you go—and save the resulting code to a file with a QBE extension. Your code will become eligible for use as a data source anywhere dBASE needs it! ❖

Printed in USA
This journal is printed on recyclable paper.